

QUEUES SIMULATOR

Horvath Ariana-Cristine

Table of Contents

1.	Assignment objective	3
2.	Analysis.....	3
2.1	Overview	3
2.2	Requirements	4
2.2.1	Functional requirements.....	4
2.2.2	Non-Functional requirements	4
2.3	Use-cases and scenarios	4
3.	Design	5
3.1	UML Diagrams	5
3.1.1	Package Diagram	5
3.1.2	Class Diagram.....	6
3.2	Packages and classes	7
3.3	GUI Design	8
3.4	Data Structures and Data Types	9
4.	Implementation	10
5.	Results.....	12
6.	Conclusions.....	18
7.	Bibliography	18

1. Assignment objective

Main objective

Design and implement a simulation application aiming to analyze queuing based systems for determining and minimizing clients' waiting time.

It should have a graphical interface through which the user can insert the simulation time, number of clients, number of queues and maximum and minimum arrival time and service time for each client.

The user should see the simulation steps in real-time.

The application should end when the simulation time ends or there are no more clients in the system and the average waiting time (how much time a client spends at the queue and in front of the queue in average), average service time and peak hour (the simulation time when there are the most clients in all queues) are displayed.

These and the log of events should be saved in a text file.

Sub-objectives

- Analyze the problem and identify requirements – how the application should work, what it should do and what are the use cases (described in chapter 2. Analysis)
- Design the polynomial calculator – which architectural pattern should be used, how the packages, classes and methods are divided (described in chapter 3. Design)
- Implement the Queues Simulator – the Java code written for the implementation (described in chapter 4. Implementation)
- Test the Queues Simulator – testing the application to check if it works properly (described in chapter 5. Results)

2. Analysis

2.1 Overview

Queues are commonly used to model real world domains. The main objective of a queue is to provide a place for a "client" to wait before receiving a "service". The management of queue-based systems is interested in minimizing the time amount their "clients" are waiting in queues before they are served.

One way to minimize the waiting time is to add more servers, i.e. more queues in the system (each queue is considered as having an associated processor) but this approach increases the costs of the service supplier.

The application should simulate (by defining a simulation time $t_{simulation}$) a series of N clients arriving for service, entering Q queues, waiting, being served and finally leaving the queues. All clients are generated when the simulation is started, and are characterized by three parameters: ID (a number between 1 and N), $t_{arrival}$ (simulation time when they are ready to go to the queue; i.e. time when the client finished shopping) and $t_{service}$ (time interval or duration needed to serve the client; i.e. waiting time when the client is in front of the queue).

The application tracks the total time spent by every client in the queues and computes the average waiting time. Each client is added to the queue with minimum waiting time when its $t_{arrival}$ time is greater than or equal to the simulation time ($t_{arrival} \geq t_{simulation}$).

The application should end when the simulation time ends or there are no more clients in the system and it displays the average waiting time (how much time a client spends at the queue and in front of the queue in average), average service time and peak hour (the simulation time when there are the most clients in all queues).

The following data should be considered as input data for the application that should be inserted by the user in the application's graphical user interface:

- Number of clients (N);
- Number of queues (Q);
- Simulation interval ($t_{simulation}$);
- Minimum and maximum arrival time ($t_{arrival}^{MIN} \leq t_{arrival} \leq t_{arrival}^{MAX}$);
- Minimum and maximum service time ($t_{service}^{MIN} \leq t_{service} \leq t_{service}^{MAX}$);

2.2 Requirements

2.2.1 Functional requirements

- The queues simulator should allow users to insert input data (all positive integers):
 - number of clients
 - number of queues
 - simulation time
 - minimum and maximum arrival time
 - minimum and maximum service time.
- The queues simulator should allow users to start the simulation
- The queues simulator should display the steps of the simulation in real time, i.e. the waiting clients and the contents of the queue
- The queues simulator should display the average waiting time: how much time a client spends at the queue and in front of the queue in average, i.e. the time from when he enters the queue until he leaves the queue
- The queues simulator should display the average service time
- The queues simulator should display the peak hour: the simulation time when there are the most clients in all queues
- The queues simulator should save the log of events and other outputs in a text file

2.2.2 Non-Functional requirements

- The queues simulator should be intuitive and easy to use by the user
- The queues simulator should not allow the user to introduce invalid input
- The queues simulator should warn the user if the input is not valid or empty

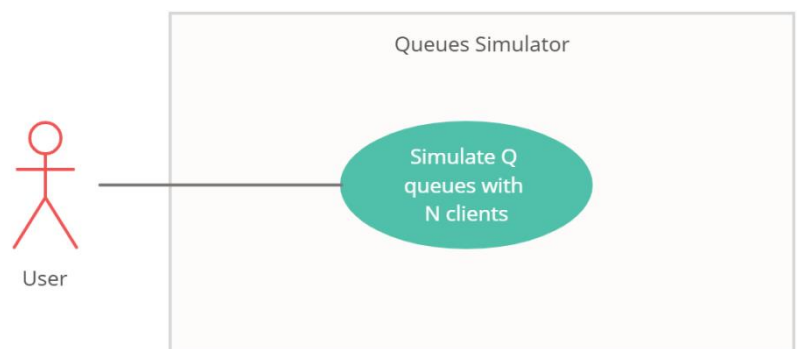
2.3 Use-cases and scenarios

Use Case: Simulate Queues

Primary Actor: user

Main Success Scenario:

1. The user inserts the input data (simulation time, number of clients, number of queues and maximum and minimum arrival time and service time) as strings in the graphical user interface, in all text fields, in the correct form (positive integers);
2. The user starts the simulation, by



pressing button “Start simulation”;

The queues simulator performs the distribution of the Q queues and displays the simulation steps in real-time (waiting clients and the contents of the queue)

4. After the simulation ends (i.e. the simulation time ends or there are no more clients in the system) the application displays the average waiting time (how much time a client spends at the queue and in front of the queue in average), average service time and peak hour (the simulation time when there are the most clients in all queues).

5. The user sees the saved log of events in the output text file.

Alternative Sequence: Incorrect input;

- The user inserts incorrect strings (not numbers), negative numbers or it leaves empty one of the text fields or more;
- A pop-up with an error message is displayed;
- The user presses the “Ok” button or “x”(Close);
- The scenario returns to step 1.

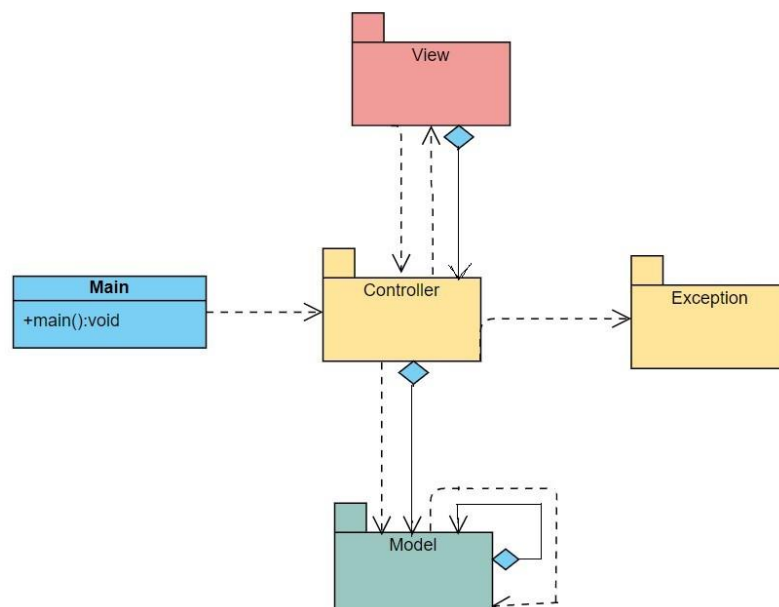
3. Design

In the design of the Queues Simulator, the MVC (Model View Controller) architectural pattern was used, which is an interactive systems pattern.

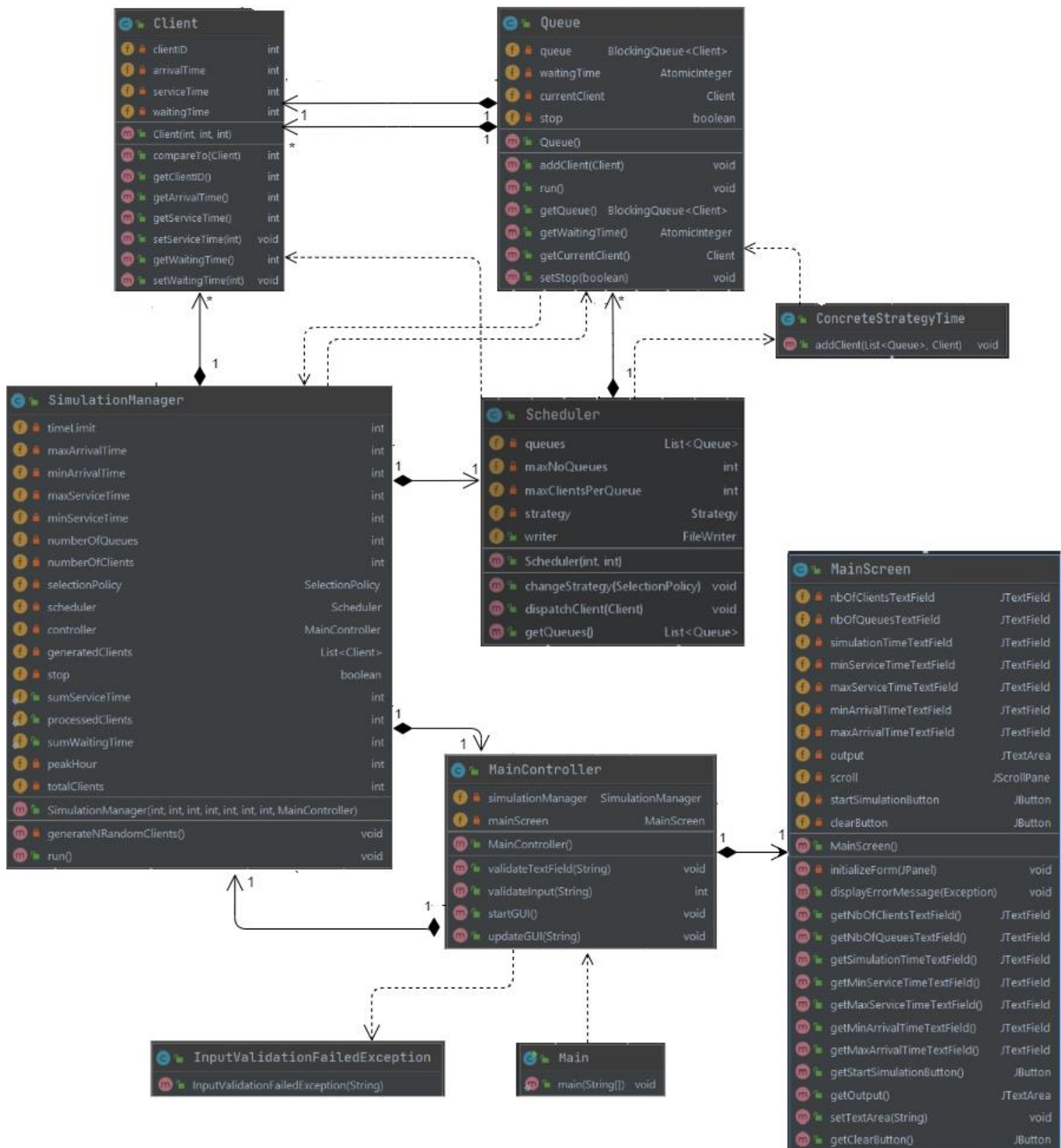
- The Model components encapsulate core data and functionality and are independent of the user interface.
- The View components receive input (which is given to the controller) and display data to the user, through the user interface.
- The Controller components make the relationship between the model and the view components, they validate and parse the input, call the methods from the model and give the output back to view components.

3.1 UML Diagrams

3.1.1 Package Diagram



3.1.2 Class Diagram



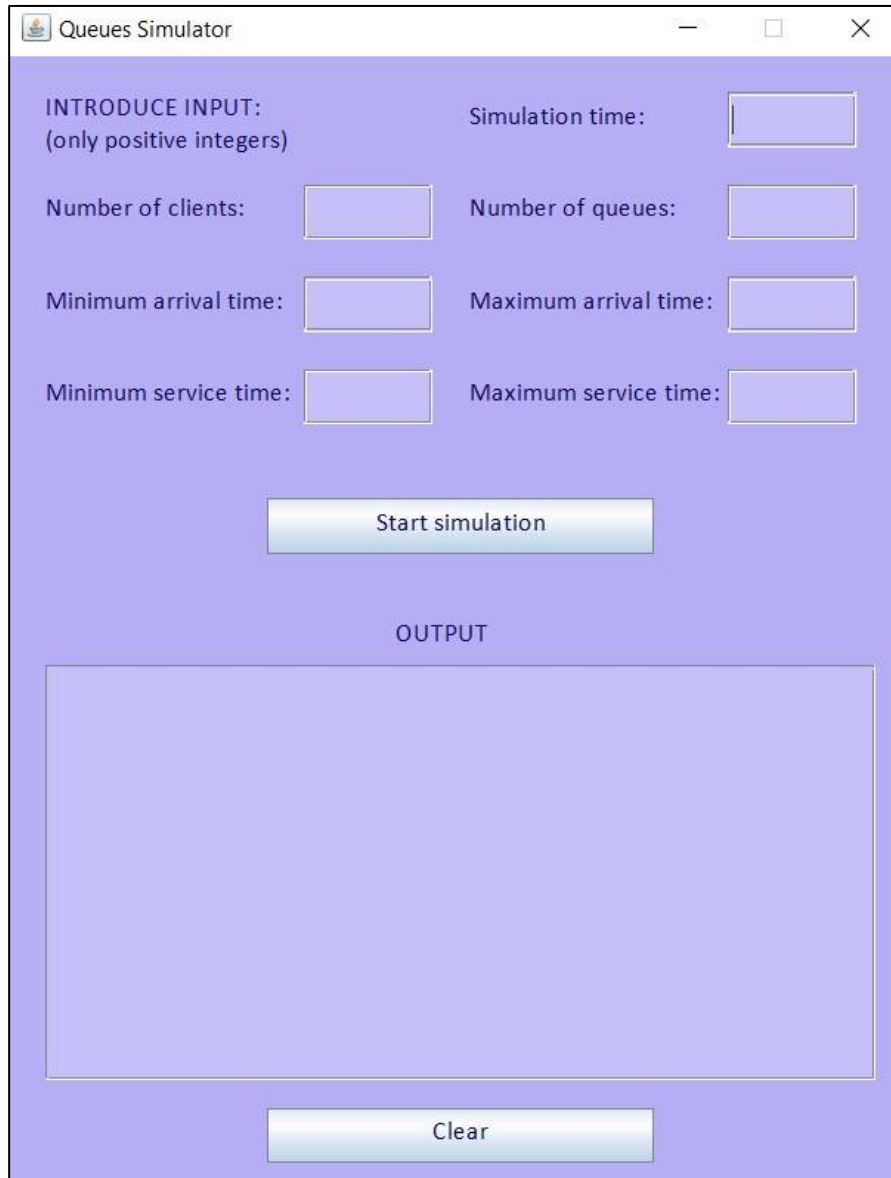
3.2 Packages and classes

-All the classes contain getters, setters and constructors, where needed.

- **Package model**
 - **Client** – implements Comparable<Client> in order to sort the clients by the arrival time
 - plays the role of a task
 - has ID, arrival time and service time, which will be randomly generated
 - **ConcreteStrategyTime** – the strategy for adding clients is by the waiting time, a new client goes in the queue where he has the least time to wait to be served
 - **Queue** – implements Runnable to define a thread
 - plays the role of a server
 - has a volatile BlockingQueue of clients and the waiting time(AtomicInteger), both used to ensure thread safety
 - **Scheduler** – sends tasks (clients) to servers (queues) according to the established strategy
 - its constructor creates and starts a thread for each queue
 - **SimulationManager** – implements Runnable
 - it is the general thread, used for simulation
 - generates N clients with arrival and service time between min and max (these will be the waiting clients)
 - its overridden method *run* does the functionality of the application, by using the Scheduler and the MainController
- **Package view**
 - **MainScreen** – extends JFrame
 - contains the code for the Graphical User Interface of the Queues Simulator having a single frame
 - the input is introduced as string in the text fields, the start button is pressed and the user sees the output displayed in real time or error messages are displayed.
- **Package controller**
 - **MainController** – it controls the data flow into model object and updates the view whenever data changes.
 - it validates the input strings and parse them into integers
- **Package exception**
 - **InputValidationFailedException** – extends RuntimeException
 - is thrown if the input is not valid
- **Main class** – instantiates the controller and thus it starts the application

3.3 GUI Design

The Graphical User Interface is very user-friendly and easy to use.



The screenshot shows a window titled "Queues Simulator". It features a light blue background. At the top left, the text "INTRODUCE INPUT: (only positive integers)" is displayed. To the right of this text is a text input field labeled "Simulation time:". Below this, there are four more input fields arranged in two rows. The first row contains "Number of clients:" and "Number of queues:". The second row contains "Minimum arrival time:" and "Maximum arrival time:". Below these, there are two more input fields: "Minimum service time:" and "Maximum service time:". A large, light blue button labeled "Start simulation" is centered below the input fields. Below the button, the word "OUTPUT" is centered. Underneath "OUTPUT" is a large, empty rectangular box for displaying output. At the bottom center of the window is a light blue button labeled "Clear".

The user introduces the input data: all fields should contain only positive integer numbers, otherwise an error message will be displayed.



Also, none of the fields can remain empty.



To start the application, the user should press the button “Start simulation” and the real time output (i.e. waiting clients and contents of the queues) will be displayed in the output text area.

After the simulation ends, either by the finishing of the simulation time or by not being any more clients in the queues or waiting, the statistics are displayed: average waiting time (how much time a client spends at the queue and in front of the queue in average), average service time and peak hour (the simulation time when there are the most clients in all queues). For these statistics, only the clients that manage to be processed are considered.

The “Clear” button clears the text from the output text area.

The user can modify the data from the fields and run the application again how many times he wants.

3.4 Data Structures and Data Types

In the implementation of my application, I used the following data structures and data types:

- **ArrayList** –for the list of Queues in SimulationManager and in Scheduler
- **ArrayBlockingQueue**^[6] – for the queue of clients in class Queue; it is a concurrent collection and here it is also volatile^[5] and used to ensure thread safety and synchronization. All queuing methods are atomic in nature and use internal locks or other forms of concurrency control.
 - BlockingQueue is a java Queue that support operations that wait for the queue to become non-empty when retrieving and removing an element, and wait for space to become available in the queue when adding an element.
- **AtomicInteger**^[4] - protects an underlying int value by providing methods that perform atomic operations on the value. It is used As an atomic counter which is being used by multiple threads concurrently, here it is used for the waiting time in class Queue.

4. Implementation

- **ConcreteStrategyTime**

- the mechanism for adding clients into a queue is done by the waiting time, a new client goes into the queue where he has the least time to wait to be served
- the index of queue with minimum waiting time is saved and the client is added in that queue, by the *addClient* method from class *Queue*
- waiting time of a client is set in order to calculate the average waiting time

```
public void addClient(List<Queue> queues, Client client) {
    int queueID = 0;
    int minTime = Integer.MAX_VALUE;

    for (Queue q : queues) {
        if (q.getWaitingTime().get() < minTime) {
            minTime = q.getWaitingTime().get();
            /** save the index of the queue with minimum waiting time */
            queueID = queues.indexOf(q);
        }
    }
    /** add the client to the queue with saved ID */
    queues.get(queueID).addClient(client);
}
```

```
public void addClient(Client client) {
    /** adds a client to the queue */
    queue.add(client);
    /** increments the waiting time with the service
    time of the added client */
    waitingTime.getAndAdd(client.getServiceTime());
    client.setWaitingTime(waitingTime.get());
}
```

- **Queue**

- the overridden method *run* from class *Queue*, which defines a thread, implements the behavior of any of the queues that run in the same time by multithreading
- they run until they are stopped from the *SimulationManager* and they wait for 1 second (through *Thread.sleep()*) for synchronization
- the first client is saved separately because *take()* will also remove it from the queue
- here are also counted the number of processed clients (because for the statistics, only the clients that get out of a queue are considered) and added the service times and waiting times for calculating the averages.

```
@Override
public void run() {
    while(!stop) {
        try {
            currentClient = queue.take(); /** save separately the current client (first from the queue) */
            int serviceTime = currentClient.getServiceTime();
            Thread.sleep( millis: currentClient.getServiceTime()*1000);
            SimulationManager.processedClients++; /** count the processed clients (they got out of the queue)*/
            SimulationManager.sumServiceTime += serviceTime;
            SimulationManager.sumWaitingTime += currentClient.getWaitingTime();
            currentClient = null;
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

- **MainController**

- initializes listeners, validates the input data, creates the simulation manager with the data and starts the application

```
public void start() {
    mainScreen.getStartSimulationButton().addActionListener(e -> {
        try {
            validateTextField(mainScreen.getNbOfClientsTextField().getText());
            int nbOfClients = validateInput(mainScreen.getNbOfClientsTextField().getText());
            validateTextField(mainScreen.getNbOfQueuesTextField().getText());
            int nbOfQueues = validateInput(mainScreen.getNbOfQueuesTextField().getText());
            validateTextField(mainScreen.getMaxArrivalTimeTextField().getText());
            int maxArrivalTime = validateInput(mainScreen.getMaxArrivalTimeTextField().getText());
            validateTextField(mainScreen.getMinArrivalTimeTextField().getText());
            int minArrivalTime = validateInput(mainScreen.getMinArrivalTimeTextField().getText());
            validateTextField(mainScreen.getMaxServiceTimeTextField().getText());
            int maxServiceTime = validateInput(mainScreen.getMaxServiceTimeTextField().getText());
            validateTextField(mainScreen.getMinServiceTimeTextField().getText());
            int minServiceTime = validateInput(mainScreen.getMinServiceTimeTextField().getText());
            validateTextField(mainScreen.getSimulationTimeTextField().getText());
            int simulationTime = validateInput(mainScreen.getSimulationTimeTextField().getText());

            simulationManager = new SimulationManager(simulationTime, maxArrivalTime, minArrivalTime,
                maxServiceTime, minServiceTime, nbOfQueues, nbOfClients, controller: this);
            Thread thread = new Thread(simulationManager);
            thread.start();
        } catch (InputValidationFailedException exception) {
            mainScreen.displayErrorMessage(exception);
        }
    });

    mainScreen.getClearButton().addActionListener(e -> {
        mainScreen.getOutput().setText("");
    });
}
```

5. Results

Test 1	Test 2	Test 3
N = 4	N = 50	N = 1000
Q = 2	Q = 5	Q = 20
$t_{simulation}^{MAX} = 60$ seconds	$t_{simulation}^{MAX} = 60$ seconds	$t_{simulation}^{MAX} = 200$ seconds
$[t_{arrival}^{MIN}, t_{arrival}^{MAX}] = [2, 30]$	$[t_{arrival}^{MIN}, t_{arrival}^{MAX}] = [2, 40]$	$[t_{arrival}^{MIN}, t_{arrival}^{MAX}] = [10, 100]$
$[t_{service}^{MIN}, t_{service}^{MAX}] = [2, 4]$	$[t_{service}^{MIN}, t_{service}^{MAX}] = [1, 7]$	$[t_{service}^{MIN}, t_{service}^{MAX}] = [3, 9]$

Test 1

Queues Simulator

INTRODUCE INPUT:
(only positive integers)

Simulation time:

Number of clients: Number of queues:

Minimum arrival time: Maximum arrival time:

Minimum service time: Maximum service time:

OUTPUT

Time 11
Waiting clients: (2, 30, 3);
Queue 1: (4, 10, 3);
Queue 2: (1, 8, 1);

Queues Simulator

INTRODUCE INPUT:
(only positive integers)

Simulation time:

Number of clients: Number of queues:

Minimum arrival time: Maximum arrival time:

Minimum service time: Maximum service time:

OUTPUT

Average waiting time: 3.25
Average service time: 3.25
Peak hour: 8

LogOfEvents.txt

```
Time 0
Waiting clients: (3, 7, 2); (1, 8, 4); (4, 10, 4); (2, 30, 3);
Queue 1: closed
Queue 2: closed

Time 1
Waiting clients: (3, 7, 2); (1, 8, 4); (4, 10, 4); (2, 30, 3);
Queue 1: closed
Queue 2: closed
```

```

Time 7
Waiting clients: (1, 8, 4); (4, 10, 4); (2, 30, 3);
Queue 1: (3, 7, 2);
Queue 2: closed

Time 8
Waiting clients: (4, 10, 4); (2, 30, 3);
Queue 1: (3, 7, 1);
Queue 2: (1, 8, 4);

Time 9
Waiting clients: (4, 10, 4); (2, 30, 3);
Queue 1: closed
Queue 2: (1, 8, 3);

Time 10
Waiting clients: (2, 30, 3);
Queue 1: (4, 10, 4);
Queue 2: (1, 8, 2);

Time 11
Waiting clients: (2, 30, 3);
Queue 1: (4, 10, 3);
Queue 2: (1, 8, 1);

Time 12
Waiting clients: (2, 30, 3);
Queue 1: (4, 10, 2);
Queue 2: closed

```

...

```

Time 30
Waiting clients:
Queue 1: (2, 30, 3);
Queue 2: closed

Time 31
Waiting clients:
Queue 1: (2, 30, 2);
Queue 2: closed

Time 32
Waiting clients:
Queue 1: (2, 30, 1);
Queue 2: closed

Time 33
Waiting clients:
Queue 1: closed
Queue 2: closed

Average waiting time: 3.25
Average service time: 3.25
Peak hour: 8

```


Test 2

Queues Simulator

INTRODUCE INPUT:
(only positive integers)

Simulation time:

Number of clients: Number of queues:

Minimum arrival time: Maximum arrival time:

Minimum service time: Maximum service time:

OUTPUT

Time 12
Waiting clients: (41, 13, 4); (21, 14, 5); (36, 15, 4); (31, 17, 1); (33, 17, 1); (7, 19, 1);
Queue 1: (20, 8, 1); (22, 12, 7);
Queue 2: (16, 12, 3);
Queue 3: (18, 7, 1); (34, 12, 6);
Queue 4: (48, 9, 5);
Queue 5: (12, 9, 2); (40, 12, 2);

Queues Simulator

INTRODUCE INPUT:
(only positive integers)

Simulation time:

Number of clients: Number of queues:

Minimum arrival time: Maximum arrival time:

Minimum service time: Maximum service time:

OUTPUT

Average waiting time: 5.58
Average service time: 3.9
Peak hour: 24

LogOfEvents.txt

```
Time 0
Waiting clients: (6, 2, 6); (47, 3, 3); (3, 6, 5); (14, 6, 1); (50, 6, 4); (18, 7, 6); (20, 8, 5); (39, 8, 1); (12, 9, 5); (48, 9, 7); (35, 11, 1);
Queue 1: closed
Queue 2: closed
Queue 3: closed
Queue 4: closed
Queue 5: closed

Time 1
Waiting clients: (6, 2, 6); (47, 3, 3); (3, 6, 5); (14, 6, 1); (50, 6, 4); (18, 7, 6); (20, 8, 5); (39, 8, 1); (12, 9, 5); (48, 9, 7); (35, 11, 1);
Queue 1: closed
Queue 2: closed
Queue 3: closed
Queue 4: closed
Queue 5: closed

Time 2
Waiting clients: (47, 3, 3); (3, 6, 5); (14, 6, 1); (50, 6, 4); (18, 7, 6); (20, 8, 5); (39, 8, 1); (12, 9, 5); (48, 9, 7); (35, 11, 1);
Queue 1: (6, 2, 6);
Queue 2: closed
Queue 3: closed
Queue 4: closed
Queue 5: closed

Time 3
Waiting clients: (3, 6, 5); (14, 6, 1); (50, 6, 4); (18, 7, 6); (20, 8, 5); (39, 8, 1); (12, 9, 5); (48, 9, 7); (35, 11, 1); (16, 12, 3);
Queue 1: (6, 2, 5);
Queue 2: (47, 3, 3);
Queue 3: closed
Queue 4: closed
Queue 5: closed
```

```

Time 13
Waiting clients: (21, 14, 5); (36, 15, 4); (31, 17, 1); (33, 17, 1); (7, 19, 5); (27, 19, 5); (28, 20, 2); (1, 21, 7); (13, 21, 2); (30, 22, 5);
Queue 1: (22, 12, 7);
Queue 2: (16, 12, 2); (41, 13, 4);
Queue 3: (34, 12, 6);
Queue 4: (48, 9, 4);
Queue 5: (12, 9, 1); (40, 12, 2);

Time 14
Waiting clients: (36, 15, 4); (31, 17, 1); (33, 17, 1); (7, 19, 5); (27, 19, 5); (28, 20, 2); (1, 21, 7); (13, 21, 2); (30, 22, 5); (32, 22, 4);
Queue 1: (22, 12, 6);
Queue 2: (16, 12, 1); (41, 13, 4);
Queue 3: (34, 12, 5);
Queue 4: (48, 9, 3);
Queue 5: (40, 12, 2); (21, 14, 5);

Time 15
Waiting clients: (31, 17, 1); (33, 17, 1); (7, 19, 5); (27, 19, 5); (28, 20, 2); (1, 21, 7); (13, 21, 2); (30, 22, 5); (32, 22, 4); (38, 22, 6);
Queue 1: (22, 12, 5);
Queue 2: (41, 13, 4);
Queue 3: (34, 12, 4);
Queue 4: (48, 9, 2); (36, 15, 4);
Queue 5: (40, 12, 1); (21, 14, 5);

Time 16
Waiting clients: (31, 17, 1); (33, 17, 1); (7, 19, 5); (27, 19, 5); (28, 20, 2); (1, 21, 7); (13, 21, 2); (30, 22, 5); (32, 22, 4); (38, 22, 6);
Queue 1: (22, 12, 4);
Queue 2: (41, 13, 3);
Queue 3: (34, 12, 3);
Queue 4: (48, 9, 1); (36, 15, 4);
Queue 5: (21, 14, 5);

```

```

Time 45
Waiting clients:
Queue 1: closed
Queue 2: (8, 38, 2);
Queue 3: closed
Queue 4: closed
Queue 5: closed

Time 46
Waiting clients:
Queue 1: closed
Queue 2: (8, 38, 1);
Queue 3: closed
Queue 4: closed
Queue 5: closed

Time 47
Waiting clients:
Queue 1: closed
Queue 2: closed
Queue 3: closed
Queue 4: closed
Queue 5: closed

Average waiting time: 5.58
Average service time: 3.9
Peak hour: 24

```

Test 3

Queues Simulator

INTRODUCE INPUT:
(only positive integers)

Simulation time:

Number of clients: Number of queues:

Minimum arrival time: Maximum arrival time:

Minimum service time: Maximum service time:

OUTPUT

Time 15
Waiting clients: (46, 16, 7); (90, 16, 9); (107, 16, 7); (277, 16, 3); (621, 16, 3);
Queue 1: (55, 10, 2); (689, 12, 4); (694, 13, 8);
Queue 2: (71, 10, 1); (494, 12, 9); (681, 14, 6);
Queue 3: (250, 10, 3); (312, 13, 8); (961, 14, 9);
Queue 4: (630, 11, 7); (695, 13, 8);
Queue 5: (436, 10, 1); (513, 12, 3); (407, 13, 8); (37, 15, 6);
Queue 6: (457, 10, 2); (86, 13, 8); (717, 14, 7);
Queue 7: (737, 11, 5); (572, 13, 8); (802, 15, 7);
Queue 8: (893, 10, 2); (198, 13, 6); (63, 14, 5); (848, 15, 8);
Queue 9: (827, 11, 3); (339, 13, 9); (79, 15, 3);

Queues Simulator

INTRODUCE INPUT:
(only positive integers)

Simulation time:

Number of clients: Number of queues:

Minimum arrival time: Maximum arrival time:

Minimum service time: Maximum service time:

OUTPUT

Average waiting time: 63.94
Average service time: 5.76
Peak hour: 100

LogOfEvents.txt

```
Time 10
Waiting clients: (13, 11, 7); (51, 11, 8); (115, 11, 4); (156, 11, 4); (382, 11, 8); (405, 11, 6); (412, 11, 5); (414, 11, 4);
Queue 1: (55, 10, 7);
Queue 2: (71, 10, 6);
Queue 3: (250, 10, 8);
Queue 4: (351, 10, 5);
Queue 5: (436, 10, 6);
Queue 6: (457, 10, 7);
Queue 7: (798, 10, 5);
Queue 8: (893, 10, 7);
Queue 9: (934, 10, 5);
Queue 10: (948, 10, 6);
Queue 11: (966, 10, 9);
Queue 12: (998, 10, 4);
Queue 13: (1000, 10, 4);
Queue 14: closed
Queue 15: closed
Queue 16: closed
Queue 17: closed
Queue 18: closed
Queue 19: closed
Queue 20: closed

Time 11
Waiting clients: (494, 12, 9); (513, 12, 3); (596, 12, 3); (656, 12, 7); (689, 12, 4); (86, 13, 8); (198, 13, 6); (221, 13, 6);
Queue 1: (55, 10, 6);
Queue 2: (71, 10, 5);
Queue 3: (250, 10, 7);
Queue 4: (351, 10, 4); (630, 11, 7);
Queue 5: (436, 10, 5);
Queue 6: (457, 10, 6);
Queue 7: (798, 10, 4); (737, 11, 5);
```


Time 90

Waiting clients: (195, 91, 9); (421, 91, 3); (504, 91, 9); (638, 91, 5); (839, 91, 3); (840, 91, 8); (875, 91, 5);
Queue 1: (334, 35, 5); (810, 36, 8); (551, 39, 9); (591, 41, 6); (831, 43, 7); (408, 46, 4); (946, 47, 3);
Queue 2: (358, 33, 3); (154, 36, 9); (108, 39, 8); (375, 41, 3); (771, 41, 5); (159, 44, 3); (877, 44, 3);
Queue 3: (228, 35, 3); (157, 36, 9); (111, 39, 6); (956, 40, 5); (889, 41, 6); (232, 44, 6); (460, 46, 6);
Queue 4: (540, 34, 2); (978, 35, 8); (531, 38, 4); (690, 39, 9); (967, 41, 7); (473, 44, 7); (865, 46, 7);
Queue 5: (187, 34, 1); (672, 35, 9); (586, 38, 8); (62, 41, 5); (227, 42, 9); (59, 45, 5); (135, 47, 9);
Queue 6: (649, 34, 5); (991, 36, 7); (117, 39, 6); (96, 41, 6); (512, 42, 3); (151, 43, 5); (133, 45, 3);
Queue 7: (601, 33, 2); (91, 36, 7); (750, 37, 8); (419, 40, 9); (908, 42, 4); (636, 44, 4); (191, 46, 9);
Queue 8: (438, 35, 3); (253, 36, 6); (820, 37, 9); (142, 41, 5); (341, 42, 7); (642, 44, 4); (244, 46, 3);
Queue 9: (554, 35, 4); (446, 36, 3); (360, 37, 4); (795, 38, 6); (617, 40, 7); (543, 42, 8); (165, 45, 7);
Queue 10: (650, 35, 8); (396, 37, 4); (139, 39, 6); (315, 41, 4); (641, 41, 5); (223, 43, 3); (651, 44, 3);
Queue 11: (469, 34, 1); (688, 35, 9); (761, 38, 6); (25, 40, 7); (378, 42, 3); (922, 42, 3); (340, 44, 6);
Queue 12: (904, 33, 4); (626, 36, 5); (939, 37, 5); (744, 39, 7); (507, 41, 9); (712, 44, 5); (727, 46, 3);
Queue 13: (15, 35, 4); (743, 36, 8); (196, 39, 3); (751, 39, 4); (322, 41, 4); (422, 42, 9); (307, 45, 5);
Queue 14: (500, 34, 1); (832, 35, 8); (520, 38, 3); (265, 39, 7); (368, 41, 6); (807, 42, 5); (719, 44, 3);
Queue 15: (101, 35, 6); (243, 37, 9); (770, 39, 7); (769, 41, 6); (170, 44, 9); (620, 47, 6); (957, 48, 4);
Queue 16: (50, 34, 3); (313, 36, 3); (310, 37, 9); (942, 39, 6); (548, 41, 6); (646, 43, 4); (5, 45, 6);
Queue 17: (122, 34, 5); (188, 37, 3); (530, 37, 5); (667, 39, 4); (764, 40, 9); (40, 43, 3); (374, 44, 5);
Queue 18: (197, 35, 5); (236, 37, 3); (574, 37, 8); (162, 40, 9); (838, 42, 7); (343, 45, 9); (230, 48, 5);
Queue 19: (301, 33, 3); (395, 36, 5); (616, 37, 9); (779, 40, 9); (112, 43, 9); (789, 46, 3); (812, 47, 3);
Queue 20: (485, 33, 2); (100, 36, 4); (321, 37, 3); (522, 38, 3); (388, 39, 5); (806, 40, 3); (480, 41, 7)

Time 91

Waiting clients: (82, 92, 5); (172, 92, 6); (186, 92, 7); (224, 92, 9); (279, 92, 9); (379, 92, 4); (416, 92, 5);
Queue 1: (334, 35, 4); (810, 36, 8); (551, 39, 9); (591, 41, 6); (831, 43, 7); (408, 46, 4); (946, 47, 3);
Queue 2: (358, 33, 2); (154, 36, 9); (108, 39, 8); (375, 41, 3); (771, 41, 5); (159, 44, 3); (877, 44, 3);
Queue 3: (228, 35, 2); (157, 36, 9); (111, 39, 6); (956, 40, 5); (889, 41, 6); (232, 44, 6); (460, 46, 6);
Queue 4: (540, 34, 1); (978, 35, 8); (531, 38, 4); (690, 39, 9); (967, 41, 7); (473, 44, 7); (865, 46, 7);

Time 199

Waiting clients:

Queue 1: (483, 67, 1); (72, 70, 5); (968, 71, 8); (700, 74, 5);
Queue 2: (231, 69, 6); (335, 72, 8); (736, 74, 3); (784, 75, 8);
Queue 3: (911, 67, 2); (217, 70, 7); (235, 73, 3); (777, 73, 9);
Queue 4: (64, 68, 2); (305, 70, 6); (1, 73, 4); (129, 74, 6); (79, 75, 5);
Queue 5: (595, 69, 7); (723, 72, 9); (14, 75, 8); (452, 77, 7);
Queue 6: (114, 68, 2); (441, 70, 6); (178, 73, 7); (905, 74, 5);
Queue 7: (615, 69, 5); (958, 71, 8); (272, 74, 6); (423, 76, 4);
Queue 8: (316, 69, 2); (585, 70, 9); (581, 73, 5); (60, 75, 3);
Queue 9: (705, 69, 3); (324, 71, 3); (459, 72, 6); (182, 74, 9);
Queue 10: (505, 67, 2); (765, 70, 7); (259, 73, 9); (833, 75, 3);
Queue 11: (418, 69, 2); (45, 71, 9); (603, 73, 3); (756, 74, 5);
Queue 12: (391, 68, 2); (97, 71, 8); (345, 73, 3); (356, 74, 3);
Queue 13: (745, 69, 2); (173, 71, 8); (470, 73, 3); (544, 74, 3);
Queue 14: (102, 69, 4); (582, 71, 3); (894, 72, 9); (528, 75, 3);
Queue 15: (632, 67, 2); (184, 71, 9); (611, 73, 3); (885, 74, 7);
Queue 16: (125, 69, 3); (361, 71, 3); (541, 72, 7); (623, 74, 6);
Queue 17: (431, 67, 1); (200, 70, 7); (225, 73, 8); (613, 75, 8);
Queue 18: (823, 69, 4); (808, 71, 7); (654, 73, 5); (762, 75, 8);
Queue 19: (979, 69, 9); (263, 73, 3); (183, 74, 9); (935, 76, 7);
Queue 20: (899, 69, 2); (222, 71, 7); (269, 73, 7); (774, 75, 9);

Average waiting time: 63.94

Average service time: 5.76

Peak hour: 100

6. Conclusions

I consider that the user interface of the Queues Simulator is intuitive and easy to use and that the application does everything that is required and it can be useful for simulating servers.

The implementation had a lot of notions to consider, mostly relating to multithreading and ensuring thread safety and synchronization.

What I learned from this assignment was a deepening of the OOP concepts learned the last semester and working with GUIs, Maven etc. but also learning new things such as implementing multithreading, using volatile and atomic variables and making the GUI updateable.

Future improvements of the application could be: a bigger GUI so that the bigger outputs can be seen better; changing the selection policy by a button (the two options are by the waiting time or by the queue size); depending on the number of queues, the GUI could create Q such queues text fields on the screen, making the output data more organized.

7. Bibliography

[1] Fundamental Programming Techniques – Lecture Slides, Assignment_2_Support_Presentation

[2] [Strategy pattern - Wikipedia](#)

[3] <https://app.diagrams.net/>

[4] <https://howtodoinjava.com/java/multi-threading/atomicinteger-example/#:~:text=The%20AtomicInteger%20class%20protects%20an,concurrent.>

[5] <http://tutorials.jenkov.com/java-concurrency/volatile.html>

[6] <https://www.journaldev.com/1034/java-blockingqueue-example/#:~:text=BlockingQueue%20is%20a%20java%20Queue,queue%20when%20adding%20an%20element.>

[7] <https://www.javacodegeeks.com/2012/12/multi-threading-in-java-swing-with-swingworker.html>